LSTM System Identification

Name: Ananda Cahyo Wibowo

NRP : 07111940000128

Undergrad Thesis Title : Data Driven Gas Lift Well And Network Optimization With Neural Network Based System Identification Using Modbus Simulator

Data Preparation

In [ ]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
#from datetime import datetime

#Read the csv file
df = pd.read_csv("upsample_min.csv")
df = pd.read_csv("upsample.csv")
df = pd.read_csv("upsampled_matlab.csv")
df2=df.drop(df.columns[0], axis=1)
data = df['glir11'].to_numpy()

split = 0.75

x1 = df['glir22'].to_numpy()[:int(split*len(data))]
x1 = x1.reshape(len(x1),1)
y1 = df['qo22'].to_numpy()[:int(split*len(data))]
y1 = y1.reshape(len(y1),1)

x2 = df['glir22'].to_numpy()[int(split*len(data)):]
y2 = df['qo22'].to_numpy()[int(split*len(data)):]
```

```python
print(f"ukuran x train: {np.shape(x1)} ukuran y train: {np.shape(y1)}")
print(f"ukuran x test: {np.shape(x2)} ukuran y test: {np.shape(y2)}")
```

```
ukuran x train: (11400, 1) ukuran y train: (11400, 1)
ukuran x test: (3801,) ukuran y test: (3801,)
```

Train Data

Preprocessing Data

In [ ]:

```python
#New dataframe with only training data
df_for_training_x = x1
df_for_training_y = y1

#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized
# normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_training_x)
scaler2 = scaler.fit(df_for_training_y)
df_for_training_scaled_x = scaler.transform(df_for_training_x)
df_for_training_scaled_y = scaler2.transform(df_for_training_y)

#As required for LSTM networks, we require to reshape an input data into n_samples x timesteps x n_features.
#In this example, the n_features is 5. We will make timesteps = 14 (past days data used for training).

#Empty lists to be populated using formatted training data
trainX = []
trainY = []

n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14  # Number of past days we want to use to predict the future.

#Reformat input data into a shape: (n_samples x timesteps x n_features)
#In my example, my df_for_training_scaled has a shape (12823, 5)
#12823 refers to the number of data points and 5 refers to the columns (multi-variables).
for i in range(n_past, len(df_for_training_scaled_x) - n_future +1):
    trainX.append(df_for_training_scaled_x[i - n_past:i, 0:df_for_training_x.shape[1]])

for i in range(n_past, len(df_for_training_scaled_y) - n_future +1):
    trainY.append(df_for_training_scaled_y[i + n_future - 1:i + n_future, 0])

trainX, trainY = np.array(trainX), np.array(trainY)
```

```
print('trainX shape == {}.'.format(trainX.shape))
print('trainY shape == {}.'.format(trainY.shape))
```

```
trainX shape == (11386, 14, 1).
trainY shape == (11386, 1).
```

RNN LSTM Architecture & Training

In [ ]:

```
# define the Autoencoder model

model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(LSTM(32, activation='relu', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(trainY.shape[1]))

model.compile(optimizer='adam', loss='mse')
model.summary("")
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 14, 64)            16896

 lstm_1 (LSTM)               (None, 14, 64)            33024

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 14, 64)            16896

 lstm_1 (LSTM)               (None, 14, 64)            33024

 lstm_2 (LSTM)               (None, 32)                12416

 dropout (Dropout)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 62,369
Trainable params: 62,369
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
# fit the model
history = model.fit(trainX, trainY, epochs=60, batch_size=15, validation_split=0.1, verbose=1)

model.save("RNN_model_resolved")
```

```
Epoch 1/60
684/684 [==============================] - 46s 60ms/step - loss: 0.5261 - val_loss: 0.1884
Epoch 2/60
684/684 [==============================] - 23s 34ms/step - loss: 0.4045 - val_loss: 0.1943
Epoch 3/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3945 - val_loss: 0.1587
Epoch 4/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3859 - val_loss: 0.1773
Epoch 5/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3944 - val_loss: 0.1913
Epoch 6/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3858 - val_loss: 0.1801
Epoch 7/60
684/684 [==============================] - 23s 33ms/step - loss: 0.3757 - val_loss: 0.2099
Epoch 8/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3856 - val_loss: 0.1531
Epoch 9/60
684/684 [==============================] - 37s 55ms/step - loss: 0.3761 - val_loss: 0.1565
Epoch 10/60
684/684 [==============================] - 48s 71ms/step - loss: 0.3714 - val_loss: 0.1825
Epoch 11/60
684/684 [==============================] - 48s 71ms/step - loss: 0.3915 - val_loss: 0.1859
Epoch 12/60
684/684 [==============================] - 35s 51ms/step - loss: 0.3741 - val_loss: 0.1761
Epoch 13/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3655 - val_loss: 0.1843
Epoch 14/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3718 - val_loss: 0.1775
Epoch 15/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3739 - val_loss: 0.1465
Epoch 16/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3677 - val_loss: 0.2210
Epoch 17/60
684/684 [==============================] - 23s 33ms/step - loss: 0.3808 - val_loss: 0.1652
Epoch 18/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3659 - val_loss: 0.1441
Epoch 19/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3634 - val_loss: 0.1723
Epoch 20/60
684/684 [==============================] - 24s 34ms/step - loss: 0.3584 - val_loss: 0.1729
Epoch 21/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3517 - val_loss: 0.1585
Epoch 22/60
```

```
684/684 [==============================] - 23s 34ms/step - loss: 0.3519 - val_loss: 0.1863
Epoch 23/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3533 - val_loss: 0.1753
Epoch 24/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3536 - val_loss: 0.1896
Epoch 25/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3572 - val_loss: 0.1912
Epoch 26/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3486 - val_loss: 0.1595
Epoch 27/60
684/684 [==============================] - 23s 34ms/step - loss: 0.3634 - val_loss: 0.1816
Epoch 28/60
684/684 [==============================] - 23s 33ms/step - loss: 0.3520 - val_loss: 0.1718
Epoch 29/60
684/684 [==============================] - 23s 33ms/step - loss: 0.3377 - val_loss: 0.1807
Epoch 30/60
684/684 [==============================] - 23s 33ms/step - loss: 0.3515 - val_loss: 0.1873
Epoch 31/60
684/684 [==============================] - 816s 1s/step - loss: 0.3424 - val_loss: 0.1917
Epoch 32/60
684/684 [==============================] - 23s 33ms/step - loss: 0.3441 - val_loss: 0.1514
Epoch 33/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3451 - val_loss: 0.1856
Epoch 34/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3474 - val_loss: 0.1494
Epoch 35/60
684/684 [==============================] - 17s 24ms/step - loss: 0.3482 - val_loss: 0.1800
Epoch 36/60
684/684 [==============================] - 17s 25ms/step - loss: 0.4103 - val_loss: 0.2386
Epoch 37/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3982 - val_loss: 0.1893
Epoch 38/60
684/684 [==============================] - 17s 24ms/step - loss: 0.4655 - val_loss: 0.2324
Epoch 39/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3616 - val_loss: 0.1763
Epoch 40/60
684/684 [==============================] - 17s 24ms/step - loss: 0.3635 - val_loss: 0.1926
Epoch 41/60
684/684 [==============================] - 17s 25ms/step - loss: 0.3555 - val_loss: 0.1694
Epoch 42/60
684/684 [==============================] - 18s 26ms/step - loss: 0.3594 - val_loss: 0.1575
Epoch 43/60
684/684 [==============================] - 17s 25ms/step - loss: 0.3636 - val_loss: 0.1657
```

```
Epoch 44/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3611 - val_loss: 0.1539
Epoch 45/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3614 - val_loss: 0.1673
Epoch 46/60
684/684 [==============================] - 16s 24ms/step - loss: 0.4644 - val_loss: 0.2967
Epoch 47/60
684/684 [==============================] - 16s 24ms/step - loss: 0.4282 - val_loss: 0.2056
Epoch 48/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3941 - val_loss: 0.2166
Epoch 49/60
684/684 [==============================] - 16s 24ms/step - loss: 0.4042 - val_loss: 0.1837
Epoch 50/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3790 - val_loss: 0.1758
Epoch 51/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3708 - val_loss: 0.1713
Epoch 52/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3852 - val_loss: 0.1851
Epoch 53/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3893 - val_loss: 0.2148
Epoch 54/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3922 - val_loss: 0.1938
Epoch 55/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3835 - val_loss: 0.1715
Epoch 56/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3579 - val_loss: 0.1815
Epoch 57/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3585 - val_loss: 0.1591
Epoch 58/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3662 - val_loss: 0.1675
Epoch 59/60
684/684 [==============================] - 16s 24ms/step - loss: 0.3942 - val_loss: 0.1539
Epoch 60/60
684/684 [==============================] - 17s 24ms/step - loss: 0.3765 - val_loss: 0.1647
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: RNN_model_resolved\assets
```

```
INFO:tensorflow:Assets written to: RNN_model_resolved\assets
```

In [ ]:

```
from tensorflow.keras.models import load_model
model = load_model("RNN_model_resolved")
```

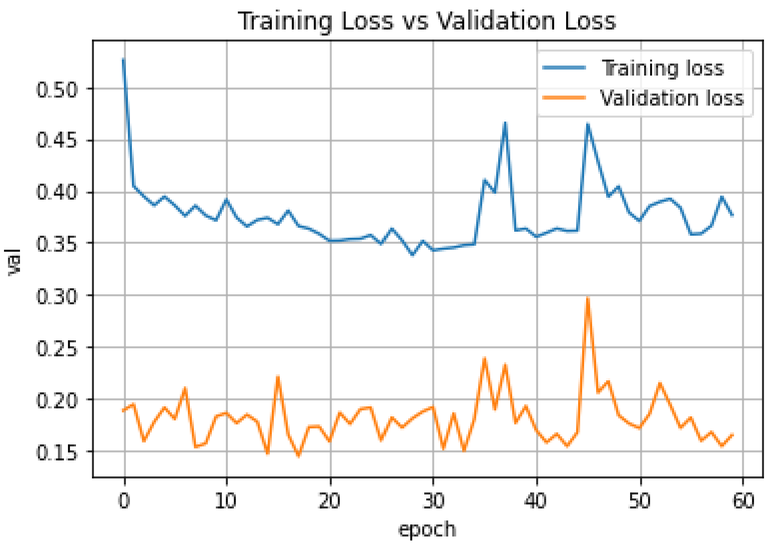
Weights and Biasses

In [ ]:
```
layers = [0,1,2,4] #layer 0:lstm 1:lstm 3:dense

weights = {}
biases = {}
for layer in layers:
    weights[layer] = model.layers[layer].get_weights()[0]
    biases[layer] = model.layers[layer].get_weights()[1]

nlayer = 1
print(np.shape(weights[nlayer]))
print(weights[nlayer])

print(np.shape(biases[nlayer]))
print(biases[nlayer])
```

```
(64, 256)
[[-0.23896053 -0.35111073 -0.14480117 ...  0.07049105 -0.51882756
  -0.23618191]
 [-0.22188431 -0.04171582 -0.11982507 ... -0.12670615  0.3324441
   0.19520585]
 [ 0.0407435  -0.20489818  0.17484163 ... -0.02960617 -0.47650638
   0.08337712]
 ...
 [-0.13235615 -0.38895538 -0.12512434 ... -0.09793747 -0.71174955
   0.00417551]
 [-0.13443659 -0.13472798  0.06700204 ...  0.09223329 -0.05157065
  -0.17175426]
 [-0.07506447 -0.3176031   0.26679012 ... -0.13549782 -0.17442518
  -0.3734845 ]]
(64, 256)
[[-0.16319828 -0.06518429  0.11444586 ... -0.03074566  0.03222966
  -0.09673079]
 [-0.04345942 -0.03886256 -0.05500256 ... -0.05092252 -0.23984362
  -0.02109332]
 [ 0.10088948 -0.08723327 -0.30343768 ... -0.07392153 -0.10041233
  -0.11727955]
 ...
 [-0.03591659  0.12921302  0.13801688 ... -0.03584264  0.03210752
  -0.03759758]
 [ 0.16241908  0.19179738  0.51935995 ... -0.16563536  0.09751325
   0.20481555]
 [ 0.13933733  0.24408627  0.00712486 ... -0.08592694 -0.44527656
  -0.11776236]]
```

In [ ]:

```
xx = np.arange(0,len(history.history['loss']))

"""print(history.history['loss'])
print(history.history['val_loss'])
print(xx)"""

plt.figure(1)
plt.plot(xx,history.history['loss'], label='Training loss')
plt.plot(xx,history.history['val_loss'], label='Validation loss')
plt.title("Training Loss vs Validation Loss")
plt.xlabel("epoch")
plt.ylabel("val")
plt.legend()
plt.grid()
```

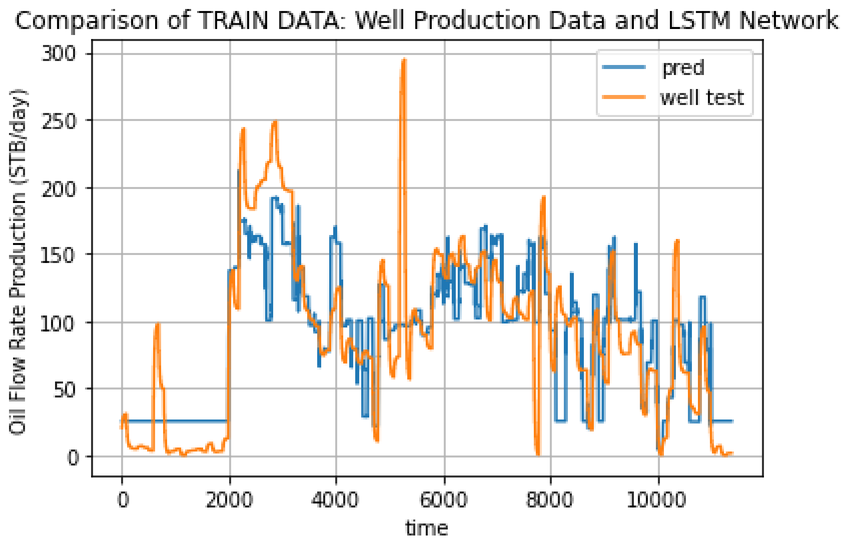


Predicting Values

In [ ]:
```
#Make prediction
#model = keras.models.load_model("RNN_Model")

n_days_for_prediction = 20
prediction = model.predict(trainX[:]) #shape = (n, 1) where n is the n_days_for_prediction

#Perform inverse transformation to rescale back to original range

prediction_copies = np.repeat(prediction, df_for_training_y.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(prediction_copies)

print('nilai pred:',y_pred_future)

yy = scaler.inverse_transform(trainY)

x_axis = np.arange(0,y_pred_future.shape[0])

print(f"ukuran y: {np.shape(yy)} ukuran y pred: {np.shape(y_pred_future)}")

plt.figure(2)
plt.plot(x_axis,y_pred_future, label='pred')
plt.plot(x_axis,yy, label='well test')
plt.title("Comparison of TRAIN DATA: Well Production Data and LSTM Network")
```

```
plt.xlabel("time")
plt.ylabel("Oil Flow Rate Production (STB/day)")
plt.legend()
plt.grid()
plt.show()
```

```
356/356 [==============================] - 3s 9ms/step
nilai pred: [[25.546812]
 [25.546812]
 [25.546812]
 ...
 [25.546812]
 [25.546812]
 [25.546812]]
ukuran y: (11386, 1) ukuran y pred: (11386, 1)
```



Metric

In [ ]:

```
import math
from sklearn.metrics import r2_score

MSE = np.square(np.subtract(yy,y_pred_future)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:")
print(RMSE)
```

```
r2 = r2_score(yy,y_pred_future)
print("\nR2 Value:")
print(r2)
```

```
Root Mean Square Error:
35.36532143265985

R2 Value:
0.6844956987192298
```

Forecasting Value/Test Data

In [ ]:

```
#df2 = pd.read_csv("upsample.csv")
#df2 = df2.iloc[:,0:152]
x2 = df['glir22'].to_numpy()[int(split*len(data)):]
y2 = df['qo22'].to_numpy()[int(split*len(data)):]

#x2 = df2['glir11'].to_numpy()
#y2 = df2['qo11'].to_numpy()

#New dataframe with only testing data
x2 = x2.reshape(len(x2),1)
y2 = y2.reshape(len(y2),1)
df_for_testing_x = x2
df_for_testing_y = y2

#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized
# normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_testing_x)
scaler2 = scaler.fit(df_for_testing_y)

df_for_testing_scaled_x = scaler.transform(df_for_testing_x)
df_for_testing_scaled_y = scaler2.transform(df_for_testing_y)

#As required for LSTM networks, we require to reshape an input data into n_samples x timesteps x n_features.
#In this example, the n_features is 5. We will make timesteps = 14 (past days data used for testing).

#Empty lists to be populated using formatted testing data
testX = []
testY = []
```

```
n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14  # Number of past days we want to use to predict the future.

#Reformat input data into a shape: (n_samples x timesteps x n_features)
#In my example, my df_for_testing_scaled has a shape (12823, 5)
#12823 refers to the number of data points and 5 refers to the columns (multi-variables).
for i in range(n_past, len(df_for_testing_scaled_x) - n_future +1):
    testX.append(df_for_testing_scaled_x[i - n_past:i, 0:df_for_testing_x.shape[1]])

for i in range(n_past, len(df_for_testing_scaled_y) - n_future +1):
    testY.append(df_for_testing_scaled_y[i + n_future - 1:i + n_future, 0])

testX, testY = np.array(testX), np.array(testY)

print('testX shape == {}.'.format(testX.shape))
print('testY shape == {}.'.format(testY.shape))
```

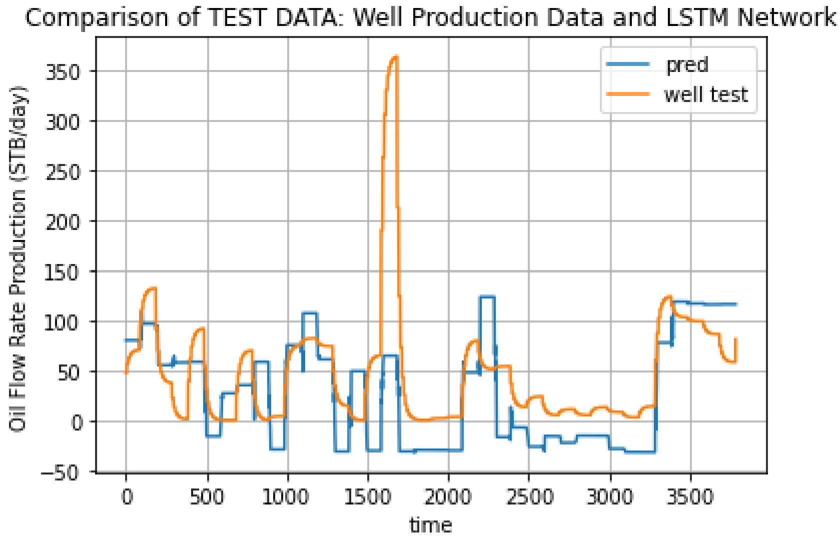
```
testX shape == (3787, 14, 1).
testY shape == (3787, 1).
```

In [ ]:

```
#Make forecast
#model = keras.models.load_model("RNN_Model")

n_days_for_forecast = 20
forecast = model.predict(testX[:]) #shape = (n, 1) where n is the n_days_for_forecast

#Perform inverse transformation to rescale back to original range

forecast_copies = np.repeat(forecast, df_for_testing_y.shape[1], axis=-1)
y_fore_future = scaler.inverse_transform(forecast_copies)

#print('nilai pred:',y_fore_future)

yyy = scaler.inverse_transform(testY)

x_axis = np.arange(0,y_fore_future.shape[0])

print(f"ukuran y: {np.shape(yyy)} ukuran y pred: {np.shape(y_fore_future)}")

plt.figure(2)
plt.plot(x_axis,y_fore_future, label='pred')
plt.plot(x_axis,yyy, label='well test')
plt.title("Comparison of TEST DATA: Well Production Data and LSTM Network")
```

```
plt.xlabel("time")
plt.ylabel("Oil Flow Rate Production (STB/day)")
plt.legend()
plt.grid()
plt.show()
```

```
119/119 [==============================] - 1s 9ms/step
ukuran y: (3787, 1) ukuran y pred: (3787, 1)
```



Metric

```
In [ ]: import math
from sklearn.metrics import r2_score

MSE = np.square(np.subtract(yyy,y_fore_future)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:")
print(RMSE)

r2 = r2_score(yyy,y_fore_future)
print("\nR2 Value:")
print(r2)
```

```
Root Mean Square Error:
58.646430603692465

R2 Value:
0.04982542715276672

58.646430603692465

R2 Value:
0.04982542715276672
```

Test in the looping

In [ ]:

```python
import random
from tensorflow.keras.models import load_model
model = load_model("RNN_model_resolved")

input = []
output = []
i = 0
n_past = 14
input_init = []
forecasting = []
while True:
    if i < n_past:
        ran = random.randint(100,700)
        input_init.append(ran)
        input_zero = np.zeros((n_past-(i+1),))
        input_zero = input_zero.tolist()

        input_total = input_init + input_zero
        input_total = np.array(input_total)
        input_total = np.reshape(input_total,(1,n_past,1))

        forecastt = model.predict(input_total) #shape = (n, 1) where n is the n_days_for_forecast
        forecastt = forecastt.tolist()[0][0]
        forecasting.append(forecastt)

        i+=1
    else:
        print("done")
        print('forecasted:',forecasting)
```

```
        fig, ax_left = plt.subplots()
        ax_left.plot(list(range(len(forecasting))),forecasting, label = 'well pred')
        ax_left.set_ylabel('well pred')

        ax_right = ax_left.twinx()
        ax_right.plot(list(range(len(input_total[0,:,:]))),input_total[0,:,:], label = 'glir')
        ax_right.set_ylabel('glir')
        i+=1
        break
```

```
1/1 [==============================] - 0s 290ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 297ms/step
done
forecasted: [-0.23220562934875488, 0.1405988484621048, -0.12677589058876038, -0.9669342041015625, -0.3909370005130768,
3.7453091144561768, 1.372929334640503, 1.645264744758606, 0.9923259019851685, 3.7705531120300293, 1.223003625869751, 6.
956583023071289, 8.06331729888916, 8.400053977966309]
```